test

Ruhan

November 2, 2015

Step1: Main Algorithmm for Beta distribution-Metropolis Hasting

```
set.seed(99)
#prepare for the traceplot in later stages:
library(coda)
```

```
## Warning: package 'coda' was built under R version 3.1.3
```

```
#Metropolis Hasting Procedure:
Beta_Metropolis <- function(alpha=6,beta=4,c,iterations){
  chain <- numeric(iterations+1)
  start_value <- runif(1)
  chain[1] <- start_value
  for(i in 1:iterations){
    current_value <- chain[i]
    #proposal function:
    newbeta <- 1
    while (newbeta == 0 | newbeta == 1) {
      newbeta <- rbeta(1, c*current_value, c*(1-current_value))
    }
    #proposal ratio:
    proposal_ratio <- dbeta(current_value, c*newbeta, c*(1-newbeta)) / dbeta(newbeta, c*current_value, c
    #posterior ratio:
    posterior_ratio <- dbeta(newbeta, alpha, beta) / dbeta(current_value, alpha, beta)
    #acceptance ratio:
    if(runif(1) < min(1, posterior_ratio * proposal_ratio)){
      chain[i+1] <- newbeta
    }else{
      chain[i+1] <- current_value
    }
  }
  return(chain)
}
```

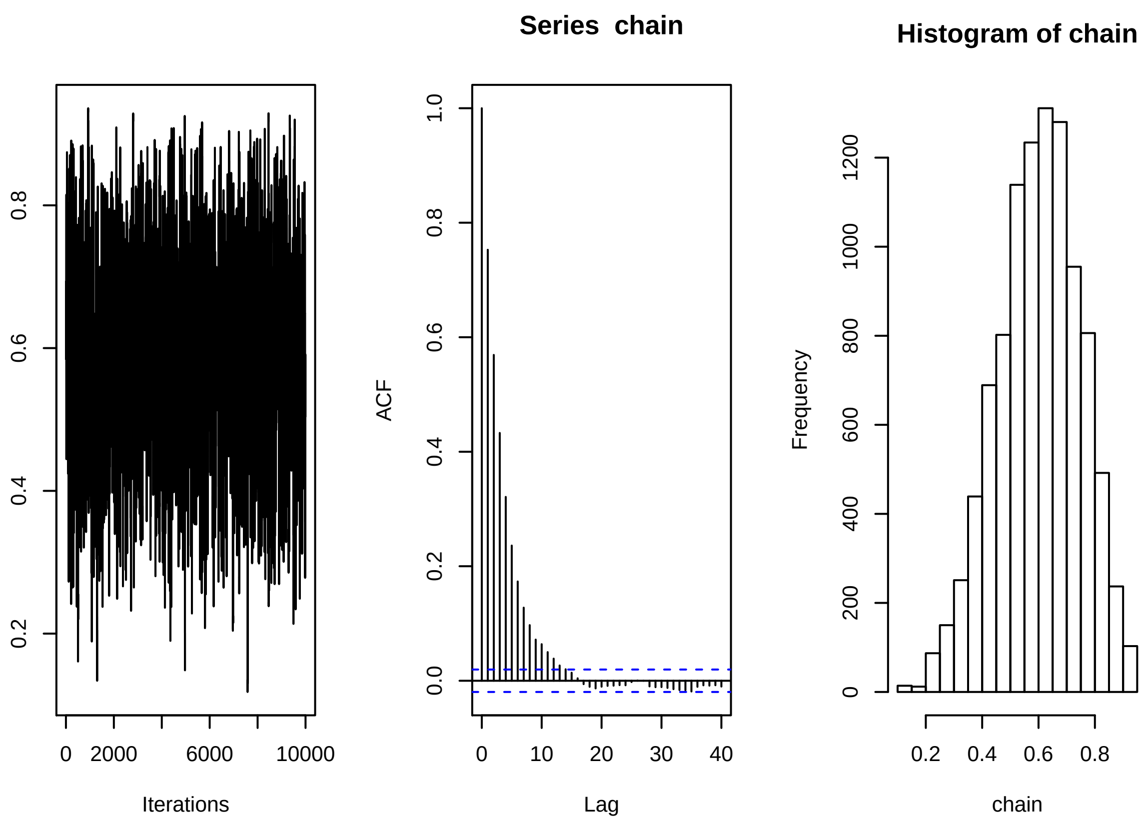
Our goal:
A Metropolis-Hastings algorithm that will potentially sample from Beta distribution(6,4).

Method:
We are constructing this algorithm under a bayesian inference platform.(In this short report, I will follow the main procedure of my algorithm)
Step1: We first randomly pick a point under Uniform(0,1) and name it x*
Step2: We are going to pick a new candidate x* from the proposal distribution(jumping distribution). The proposal function Q(x) is defined as: ϕ_{prop} | ϕ_{old} ~ Beta(cϕ_{old}, c(1-ϕ_{old}))
Step3: We are going to calculate the acceptance rate, which can be broke into two parts: 'proposal ratio' and 'jumping ratio'. (Since Beta is not a symmetric distribution, we will have to include a jumping ratio into the formula)
Proposal ratio is defined as: Q(x|x) / Q (x|x)

Posterior ratio is defined as: P(x) / P(x*)*
*Acceptance ratio is calculated as min(1, proposal ratio*posterior ratio). If this ratio is larger than a random number generated under uniform(0,1), we will accept this new point as out next x. If this ratio is smllaer than the random number generated under uniform(0,1), we will reject this new points. Instead, we will set the next x having the same value as the current x.

Step2: evaluate the performance of the sampler

```r
chain <- Beta_Metropolis(alpha=6,beta=4,c=1,iterations=10000)
par(mfrow=c(1,3))  #1 row, 3 columns
traceplot(as.mcmc(chain)); acf(chain); hist(chain)  #plot commands
```

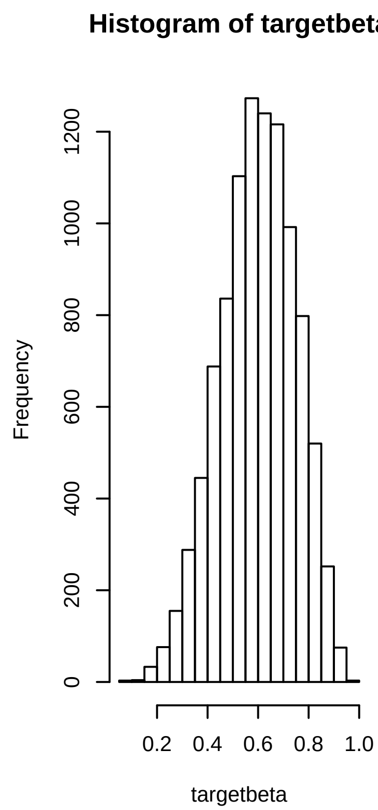



```r
#graphical comparison
targetbeta <- rbeta(10000,6,4)
hist(targetbeta)
#numerical comparison - Kolmogorov-Smirnov statistic
ks.test(chain,targetbeta)
```

```
## Warning in ks.test(chain, targetbeta): p-value will be approximate in the
## presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  chain and targetbeta
```

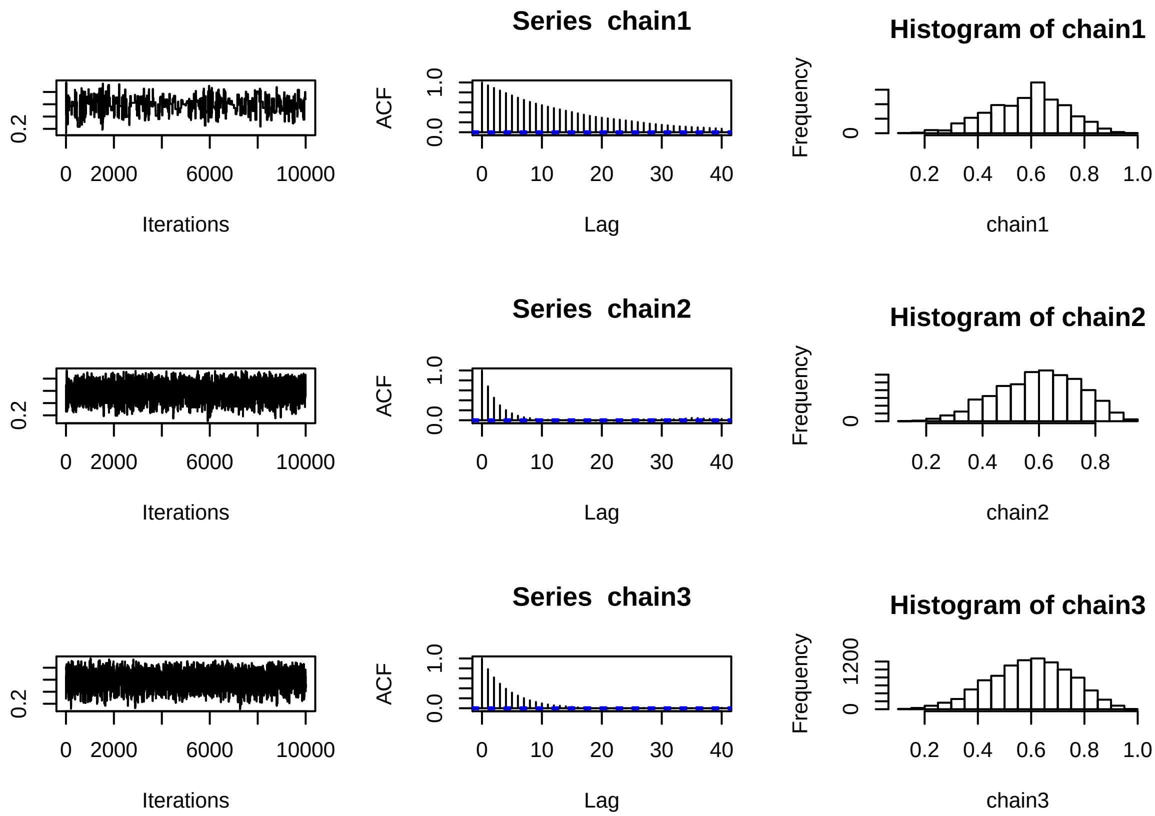
```
## D = 0.0193, p-value = 0.04914
## alternative hypothesis: two-sided
```



Graphical Method: Comparing the histogram of the chain and the histrogram generated from the target Beta distribution, we find that these two histograms look very similar, suggesting that the Metropolis-Hastings could get samples from the target Beta distribution very well. Numerical Method: Using Kolmogorov-Smirnov test, we are trying to test the if our chain follows our target Beta distribution, Beta('alpha'=6, 'beta'=4). The null hypothesis of this test states that our chain follows the target Beta distribution. The p-value for this test is 0.04914<0.05 if we are evaluating at 95% significance level. Therefore, we do not reject the null hypothesis. Our Metropolis-Hastings algorithm performs quite good in getting samples from the target Beta distribution.

Step3: re-run the sampler with c=0.1, c=2.5 and c=10

```
par(mfrow=c(3,3))
chain1 <- Beta_Metropolis(alpha=6,beta=4,c=0.1,iterations=10000)
traceplot(as.mcmc(chain1)); acf(chain1); hist(chain1)
chain2 <- Beta_Metropolis(alpha=6,beta=4,c=2.5,iterations=10000)
traceplot(as.mcmc(chain2)); acf(chain2); hist(chain2)
chain3 <- Beta_Metropolis(alpha=6,beta=4,c=10,iterations=10000)
traceplot(as.mcmc(chain3)); acf(chain3); hist(chain3)
```



We are comparing the performance of the Metropolis-Hastings algorithm under different c values where c=0.1/2.5/10.
Comparing three autocorrelation graphs, we can see that the alogorithm performs the worst under c=0.1 since the autocorrelation is still very high at large lags. When c=0.1, a higher extent of thinning is clearly required, meaning that we should increase the number of burn-in draws. c=10 has a similar issue, though to a lesser degree. It also requires some degrees of thinning, however, the number of burn-in draws in this case should be less than that in c=0.1 Thus, c=2.5 is most effective at drawing from the target Beta distribution because it has lower autocorrelation at higher lags and its histogram resembles the target Beta histogram the most.